

# State transition algorithm for traveling salesman problem

YANG Chunhua<sup>1</sup>, TANG Xiaolin<sup>1</sup>, ZHOU XiaoJun<sup>1,2</sup>, GUI Weihua<sup>1</sup>

1. School of Information Science and Engineering, Central South University, Changsha 410083, P. R. China

2. School of Science, Information Technology and Engineering, University of Ballarat, Victoria 3353, Australia

E-mail: tiezhongyu2010@gmail.com

**Abstract:** Discrete version of state transition algorithm is proposed in order to solve the traveling salesman problem. Three special operators for discrete optimization problem named swap, shift and symmetry transformations are presented. Convergence analysis and time complexity of the algorithm are also considered. To make the algorithm simple and efficient, no parameter adjusting is suggested in current version. Experiments are carried out to test the performance of the strategy, and comparisons with simulated annealing and ant colony optimization have demonstrated the effectiveness of the proposed algorithm. The results also show that the discrete state transition algorithm consumes much less time and has better search ability than its counterparts, which indicates that state transition algorithm is with strong adaptability.

**Key Words:** State Transition Algorithm, Traveling Salesman Problem, Swap, Shift, Symmetry

## 1 Introduction

As one of the most significant combinatorial optimization problems, traveling salesman problem (TSP) has been paid greatly attention and been extensively studied, due to its importance in manufacturing, distribution management and scheduling[1, 2]. Traveling salesman problem can be described as: Given a set of  $n$  nodes and distances for each pair of nodes, find a roundtrip of minimal total length visiting each node exactly once, and according to whether the distance from node  $i$  to node  $j$  is the same as from node  $j$  to node  $i$ , the TSP can be symmetric or asymmetric[3]. For a  $n$  nodes asymmetric TSP, there exists  $(n-1)!$  possible solutions; while for the corresponding symmetric TSP,  $(n-1)!/2$  solutions are possible. Because of its NP-hard property, emphasis has shifted from the aim of finding a global optimal solution to the goal of obtaining “good solutions” in reasonable time and establishing the “the degree of goodness”[4, 5].

Taking the impractical exhaustive search for TSP into consideration, heuristic algorithms are introduced to speed up the process of finding a satisfactory solution, of which, simulated annealing (SA), tabu search (TS), genetic algorithm (GA), ant colony optimization (ACO), particle swarm optimization (PSO) have found widest applications in the the field[6–10]. In terms of the concepts of state and state transition, a new heuristic search algorithm named state transition algorithm (STA) is proposed recently, and it exhibits good search performance in continuous space[11, 12]. In the initial version of state transition algorithm, a special transformation named general elementary matrix was also proposed to solve the discrete optimization problem. In this paper, we will continue to develop its detailed forms to promote deep study of STA for solving traveling salesman problem.

The paper is organized as follows: In section 2, we review

the unified form of the state transition algorithm, and establish the discrete version of STA. Then detailed forms and implementations of three special transformation operators are put forward in section 3, and we also give some strategies of parameters setting. In the next section, some experiments are performed to test the performance of the proposed algorithm. Conclusion is derived in the end.

## 2 State transition algorithm in discrete version

As stated in[11], a solution to a specific optimization problem can be described as a state, the thought of optimization algorithms can be treated as state transition, and the process to update solutions will become a state transition process. Without loss of generality, the unified form of state transition algorithm can be described as

$$\begin{cases} x_{k+1} = A_k x_k + B_k u_k \\ y_{k+1} = f(x_{k+1}) \end{cases} \quad (1)$$

where  $x_k \in \mathbb{R}^n$  stands for a state, corresponding to a solution of a optimization problem;  $A_k, B_k \in \mathbb{R}^{n \times n}$  are state transition matrixes, which are usually transformation operators;  $u_k \in \mathbb{R}^n$  is the function with variables  $x_k$  and history states;  $f$  is the cost function or evaluation function.

As for traveling salesman problem, the cost  $f$  is usually expressed as the function of a sequence, which corresponds to an ordered traveling route. That's to say, after a transformation using  $A_k$  or  $B_k$ , a new state  $x_{k+1}$  should be a sequence too. Due to the particularity of TSP, only a state transition matrix is considered, avoiding the complexity of “adding” one sequence to another.

According to the theory of Linear Algebra, we know that the special matrix must have only one position with value 1 in each column and each row. The special random matrix is called general elementary matrix, because it is derived from identity matrix and has the function of transforming a sequence into another one. For simplicity and specificity,  $G_k$  is denoted in state transition algorithm for discrete optimization problem as

YANG Chunhua's work is supported by the National Science Found for Distinguished Young Scholars of China (Grant No. 61025015), and ZHOU XiaoJun is supported by China Scholarship Council, who is the corresponding author of this paper.

follows

$$\begin{cases} x_{k+1} = G_k x_k \\ y_{k+1} = f(x_{k+1}) \end{cases} \quad (2)$$

where  $x_k = [x_{1k}, x_{ik}, \dots, x_{nk}]^T$ ,  $x_{ik} \in \{1, 2, \dots, n\}$  be a sequence, the  $G_k$  is general elementary matrix with only position value 1 in each column and each row. For instance,  $G_k$  has the following styles[11]

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

If an initial sequence is [1,2,3,4,5], then after the above general elementary matrices, it will becomes [1,2,5,4,3], [4,2,3,5,1], [4,2,1,5,3], respectively.

### 3 State transition operators and corresponding adjusting strategies

As can be seen from above, the general elementary matrix can have various types, and arbitrary types may have neither validity nor efficiency to discrete optimization problem. To make the state transition process controllable and efficient, three special transformation operators for traveling salesman problem are defined.

#### (1) Swap transformation

$$x_{k+1} = G_k^{swap}(m_a)x_k \quad (3)$$

where  $G_k^{swap} \in \mathbb{R}^{n \times n}$  is called swap transformation matrix,  $m_a$  is a constant called swap factor to control the maximum number of random positions to be exchanged, while the positions are random. For example, if  $n = 5$ ,  $m_a = 3$ , the swap transformation matrix has the following styles

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

If an initial sequence is [1,2,3,4,5], then after the above swap transformation matrices, it will becomes [3,2,1,4,5], [1,3,2,4,5], [1,5,3,2,4], respectively. It can be found that the 1<sup>th</sup> and 3<sup>th</sup> rows are exchanged, the 2<sup>th</sup> and 3<sup>th</sup> rows are exchanged, and the 2<sup>th</sup>, 4<sup>th</sup>, and 5<sup>th</sup> rows are interchanged, respectively, which indicates that the maximum number of 3 positions are exchanged.

#### (2) Shift transformation

$$x_{k+1} = G_k^{shift}(m_b)x_k \quad (4)$$

where  $G_k^{shift} \in \mathbb{R}^{n \times n}$  is called shift transformation matrix,  $m_b$  is a constant called shift factor to control the maximum length of consecutive positions to be shifted. By the way, the selected position to be shifted after and positions to be shifted are chosen randomly. To make it more clearly, let  $n = 5$ ,  $m_b = 2$ , then the shift transformation matrix has the following styles

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

If an initial sequence is [1,2,3,4,5], then after the above shift transformation matrices, it will becomes [1,3,2,4,5],

[1,3,4,2,5], [1,4,2,3,5], respectively. In the first two cases, the position to be shifted is {2}, and the positions to be shifted after are {3}, {4}, while in the third case, the positions to be shifted is {2,3}, and the position to be shifted after are {4}. In the last case, {2,3} are the maximum length of consecutive positions to be shifted.

#### (3) Symmetry transformation

$$x_{k+1} = G_k^{sym}(m_c)x_k \quad (5)$$

where  $G_k^{sym} \in \mathbb{R}^{n \times n}$  is called symmetry transformation matrix,  $m_c$  is a constant called symmetry factor to control the maximum length of subsequent positions as center. By the way, the component before the subsequent positions and consecutive positions to be symmetrized are both created randomly. For instance,  $n = 5$ ,  $m_c = 1$ , then the symmetry transformation matrix has the following styles

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

If an initial sequence is [1,2,3,4,5], then after the above symmetry transformation matrices, it will becomes [1,2,4,3,5], [1,5,4,3,2], [1,2,5,4,3], respectively. All of the three cases, the component before the subsequent positions is {3}. In the first two cases, the subsequent position is  $\{\emptyset\}$ , while the consecutive positions to be symmetrized are {4} and {4,5}, respectively. In the third case, the subsequent position(or the center) is {4}, while the consecutive position to be symmetrized is {5}. It is not difficult to find that the length of subsequent positions is 1 in the last case, which is the maximum length of subsequent positions as indicated by  $m_c$ .

To accept a new solution, “greedy criterion” is commonly adopted, in other words, only solution better than previous one is accepted. However, in SA, a worse solution is accepted probabilistically. Although the strategy in SA can jump out of stagnant state and it creates a big perturbation to previous solution, to guarantee the convergence of SA becomes difficult. In current version of state transition algorithm for discrete optimization problem, “greedy criterion” is inherited. The core procedure of the STA for TSP can be outlined in pseudocode as follows

```

1: repeat
2:   [Best,fBest] ← op_swap(cities,Best,fBest,SE,n,m_a)
3:   [Best,fBest] ← op_shift(cities,Best,fBest,SE,n,m_b)
4:   [Best,fBest] ← op_symmetry(cities,Best,fBest,SE,n,m_c)
5: until the specified termination criterion is met

```

where, *cities* is the information about the TSP; the *SE* is the times of transformation, called search enforcement. As for more detailed explanations, op\_swap function of above pseudocode is also given in MATLAB scripts

```

for i=1:SE
    Tranf = swap_matrix(n,ma);
    State(i,:) = (Tranf*Best)';
end
[newBest,fBest] = fitness(State,cities,SE);
if fBest < fBest
    Best = newBest;

```

```

fBest = fGBest;
end
function y = swap_matrix(n,ma)
y = eye(n);
R = randperm(n);
T = R(1:ma);
S = T(randperm(ma));
y(T,:) = y(S,:);

```

#### 4 Theoretical analysis of discrete STA

Next, we analyze the convergence performance, the global search ability, and time complexity of the discrete state transition algorithm.

##### 4.1 Convergence properties of discrete STA

Firstly, we define the concept of convergence for discrete STA

$$|f(x_k) - f(x^*)| \leq \epsilon, \forall k > N \quad (6)$$

while  $x^*$  is the global minimum solution of a traveling salesman problem,  $\epsilon$  is a small constant, and  $N$  is a natural number. If  $\epsilon > 0$ , we can say that the algorithm converges to a  $\epsilon$ -optimal solution; if  $\epsilon = 0$ , we can say that the algorithm converges to a global minimum. However, if  $x^*$  is a local minimum solution, then we can say that the algorithm converges to a  $\epsilon$ -suboptimal solution and a local minimum for  $\epsilon > 0$  and  $\epsilon = 0$ , respectively.

**Theorem 1** the discrete STA can at least converge to a local minimum.

*Proof.* Let suppose the maximum number of iterations (denoted by  $M$ ) is big enough, then there must exist a number  $N < M$ , when  $k > N$ , no update of the better solution will happen. That is to say,  $f(x_k) = f(x^{best}), \forall k > N$ , where  $x^{best}$  is the solution in the  $N$ th iteration. The  $x^{best}$  is just the local minimum solution as  $x^*$ , and  $|f(x_k) - f(x^{best})| = 0$ .

##### 4.2 Global search ability of discrete STA

It is not difficult to find that whether the discrete STA converges to a global minimum depends on the  $x^{best}$  in the  $N$ th iteration. If the  $x^{best}$  is the global minimum, then according to the “greedy criterion” we use to update the *best* in pseudocode, when  $k > N$ , *best* will always be  $x^{best}$ . In other words, the global convergence performance has much to do with the three operators we have designed.

**Theorem 2** the discrete STA can converge to a global minimum in probability.

*Proof.* Let suppose  $x^* = (a_1, \dots, a_n)$  is the global minimum solution, and  $x_k = (b_1, \dots, b_n)$  is the  $k$ th best solution. Then we discuss separately that how the state  $x_k$  can transform to the best state  $x^*$ .

If there exists the same sub-sequence in both  $x_k$  and  $x^*$ , for instance,  $x_k = (a_n, a_2, \dots, a_{n-1}, a_1)$ , then both swap transformation with small swap factor and appropriate shift transformation have the probability to swap or shift other positions in the sequence so that other positions in  $x_k$  can transform exactly to the corresponding positions in the  $x^*$ .

If there exists no same sub-sequence in  $x_k$  and  $x^*$ , for instance,  $x_k = (a_{n-1}, a_{n-2}, \dots, a_2, a_1)$ , then swap transformation with big swap factor and proper symmetry transformation have the probability to swap or symmetrize all positions in the sequence so that all positions in  $x_k$  can transform exactly to the corresponding positions in the  $x^*$ .

##### 4.3 Time complexity of discrete STA

Because of the NP-hard property of TSP, it is impossible to solve the problem in polynomial time. As described in the section 1, to obtain a “good solution” in a reasonable time is a optional choice, and this is the same case as the discrete STA, which aims at getting a satisfactory solution in as short a time as possible. In the pseudocode as described above, we can find that in the out loop, there are  $M$  iterations, and in the inner loop, there exist three times of  $SE$  transformations. It is not difficult to find that the time complexity of the discrete STA is  $O(M \cdot SE)$ , that is to say, the discrete STA can achieve a global optimum in polynomial time in probability.

#### 5 Experimental results and discussion

From the MATLAB scripts in section 3, we can find that as the  $m_\alpha$  increases, the exchanging transformation matrix will become less efficient. The same phenomenon can be observed in other two transformations, and repetition of the transformation matrices appears frequently.

For simplicity and efficiency, in current version of STA for TSP, parameters for three transformation operators are made as smaller as possible, that is to say,  $m_a = 2, m_b = 1, m_c = 0$  are specified in the paper, to avoid the trouble of parameters’ adjusting.

In order to test the performance of the proposed STA in discrete version, some benchmark symmetric traveling salesman problems from [3] are utilized for experiment, including *ulysses16.tsp*, *att48.tsp* and *berlin52.tsp*. In the same time, SA, ACO, which are recognized as distinguished algorithms for TSP are used for comparison with STA. In SA, we set initial temperature at 5000, cooling rate at 0.97, and in ACO,  $\alpha = 1, \beta = 5, \rho = 0.9$ , where,  $\alpha, \beta$  are used to control the relative weight of pheromone trail and heuristic value, and  $\rho$  is the pheromone trail decay coefficient. In ACO and STA, the number of ants or the search enforcement is 20, and maximum iterations is fixed at 200. Considering that SA is not usually population based algorithm, the maximum iteration is extended especially for fairness. The threshold, or the total number of iterations in SA is set at 4000.

Programs are run independently for 20 trails for each algorithms in MATLAB R2010b (version of 7.11.0.584) on Intel(R) Core(TM) i3-2310M CPU @2.10GHz under Window 7 environment, and comparison results for STA with SA and ACO are listed in Table 1. Some statistics and the run time are utilized to evaluate the performance of algorithms. The *best* means the minimum of the results, the *worst* indicates the maximum, and then it follows the *mean*, *st.dev.* (standard deviation). The run time is the average time used in 20 trails, which is measured in seconds.

As can be seen from the Table, STA outperforms SA and

Table 1: Results for benchmark test problems

Problems	Performance	SA	ACO	STA
ulysses16.tsp	best	73.9998	74.6287	73.9876
	mean	74.4481	76.0864	74.0779
	worse	75.5391	78.7728	74.5939
	st.dev.	0.4105	1.1062	0.1626
	time(s)	2.9975	11.3038	1.2223
att48.tsp	best	3.5266e4	3.7015e4	3.3724e4
	mean	3.9667e4	3.8449e4	3.4872e4
	worse	4.5887e4	3.9801e4	3.6205e4
	st.dev.	2.7453e3	862.4546	668.7553
	time(s)	14.7605	102.4784	3.0462
berlin52.tsp	best	8.1864e3	8.2404e3	7.5444e3
	mean	8.9838e3	8.7776e3	8.2472e3
	worse	9.5858e3	9.1513e3	8.6305e3
	st.dev.	380.1004	267.1124	273.4509
	time(s)	139.8399	118.0948	3.3438

ACO in almost every performance index. To be more specific, the results of each benchmark problem are discussed separately in the following.

*ulysses16.tsp*: The *best* of the results is obtained by STA, with the sequence of (7 6 14 13 12 16 1 3 2 4 8 15 5 11 9 10), which gets the best length of route at 73.9876, and the best route is plotted in Fig.1. To be more careful, we can find that the *worst* solution gained by STA is even better than the *best* of ACO, which indicates the strong search capability of STA. The *st.dev.* of STA is almost approaching zero, and it shows that STA is also stable for this test problem. By the way, the STA consumes the half time of SA, and 1/9 time of ACO. The curves of the average fitness are illustrated following in Fig.2. The data created by SA are condensed to the same iterations (the same method applied to other two results obtained by SA). It is interesting to find that the ups and downs of the curve of SA in the early stage, because SA accepts a relatively worse solution by probability. Take the condensed data of SA into consideration, we can perceive that SA needs quite a long time for its steadily descending trend. However for ACO, after quickly to each a good fitness, it is trapped into stagnation point. But for STA, neither of the phenomena occurs, and it keeps decreasing before not a short time.

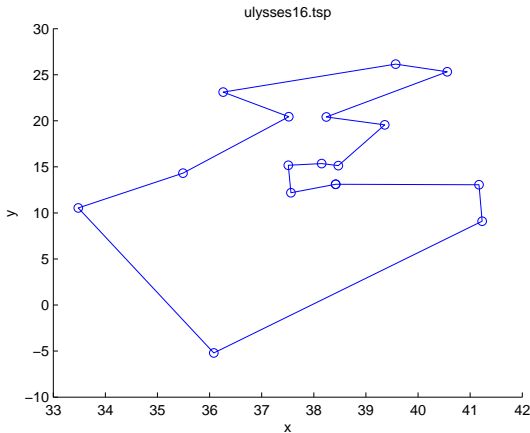


Fig. 1: the best route of ulysses16.tsp obtained by STA

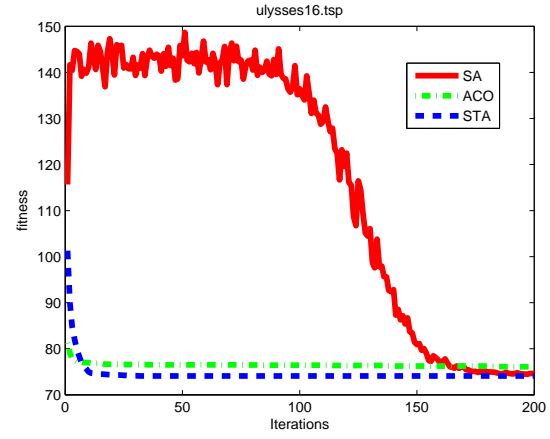


Fig. 2: curve of the average fitness for ulysses16.tsp

*att48.tsp*: STA also achieves the *best*, with the sequence of (9 40 15 12 11 23 3 22 16 41 34 48 5 29 2 42 26 4 35 45 10 24 32 39 25 14 13 21 47 20 33 46 36 30 43 17 27 19 37 6 28 7 18 44 31 38 8 1) and length of route at 3.3724e4, which is illustrated in Fig.3. The same situation is observed in the results between ACO and STA, that is, the *worst* solution gained by STA is better than the *best* of ACO. SA outperforms ACO in the *best*, but *mean* and *worst* in ACO are better than that of SA. In this case, the *st.dev.* of STA is not very satisfactory, although it is better than its counterparts. The time costed by STA is much shorter than other two algorithms, only around 1/4 and 1/30 time of that of SA and ACO consume, respectively. From Fig.4, we can find low degree of ups and downs in the curve of SA, and SA keeps good descending fitness in the first stage. ACO confronts the premature convergence again, few changes have happened in the late process. For STA, the fitness decrease sharply in the early stage, but can still keep decreasing in the later, which indicates the excellent performance of the designed operators in the discrete STA.

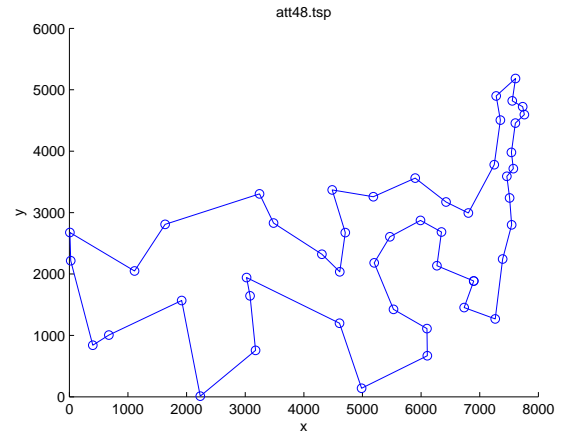


Fig. 3: the best route of att48.tsp obtained by STA

*berlin52.tsp*: The STA wins the *best* again, with the best sequence of (3 17 21 42 7 2 30 23 20 50 29 16 46 44 34 35

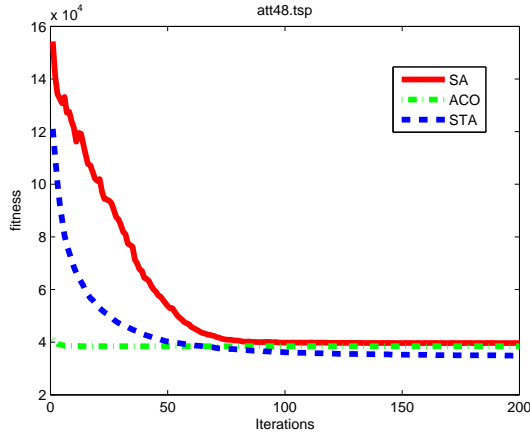


Fig. 4: curve of the average fitness for att48.tsp

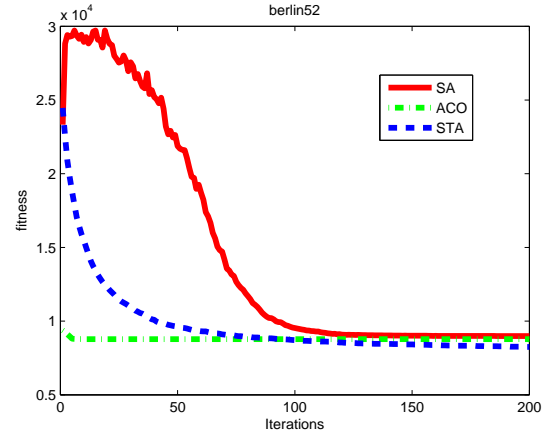


Fig. 6: curve of the average fitness for berlin52.tsp

36 39 40 37 38 48 24 5 15 6 4 25 12 28 27 26 47 13 14 52 11 51 33 43 10 9 8 41 19 45 32 49 1 22 31 18) and length of route at  $7.5444e3$ , which can be observed in Fig.5. At this time, ACO exhibits much better than SA in other performance except for the *best*. But, STA achieves the best results on the whole, especially for the computational time. For the problem, STA consumes respectively 1/40 and 1/35 of the time costed by SA and ACO. As for TSP, the time complexity is really important, so the results gained by STA shows again that the discrete STA is really promising. In Fig.6, we can find that SA need quite a long time to reach a relatively good fitness, and then it becomes stagnated. On the contrary, the fitness of STA can decrease quickly and keep descending till the end of the process.

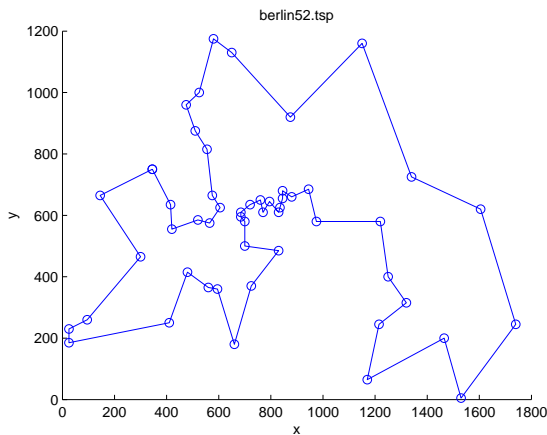


Fig. 5: the best route of berlin52.tsp obtained by STA

## 6 Conclusion

Different from continuous search space, the space for traveling salesman problem is discrete, which corresponds to a permutation of a sequence. In discrete version of state transition algorithm, three special state transition operators are designed to manipulate the permutation. A simplest but much efficient

parameters control strategy is proposed, in which, no parameters of the operators need adjusting. Some experiments are done to evaluate the proposed algorithm, and the results show that the discrete version of STA has much better performance not only in the search ability but also in the time consuming.

Premature phenomenon is extensively existing in heuristic algorithms. To escape from a stagnation point, the paper focuses on the designing of operators. Accepting a relatively worse solution is a good idea, as seen in SA; however, it increases the computing time and risks the non-convergence. It is really excited to see the fantastic performance of discrete STA only with “greedy criterion”, due to the excellent operators designed. On the other hand, difficulties are still to be confronted with large size problem, and the adjusting of the parameters need reconsidering. By the way, in current version of STA, three transformations are only for local permutation, and effective global permutation will be found in our future work as well as the equilibrium between them.

## References

- [1] Stuetzle, T., The traveling salesman problem: state of the art, in *TUD-SAP AG Workshop on Vehicle Routing*, July 10, 2003.
- [2] G Laporte, A concise guide to the traveling salesman problem, *Journal of the Operational Research Society*, 61(2010), 35–40.
- [3] <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/>
- [4] C.H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Dover Publications, Inc., New York, 1982.
- [5] Zakir H. Ahmed, Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator, *International journal of biometrics & bioinformatics*, 3(6): 96–105, 2010.
- [6] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by simulated annealing, *Science*, 220(4590): 671–680, 1983.
- [7] John Knox, Tabu search performance on the symmetric traveling salesman problem, *Computers & operational research*, 21(8): 867–876, 1994.
- [8] Jinhui Yang, Chunguo Wua, Heow Pueh Lee, Yanchun Liang, Solving traveling salesman problems using generalized chromosome genetic algorithm, *Progress in natural science*, 18(2008): 887–892.

- [9] Marco Dorigo, Luca Maria Gambardella, Ant Colony System: A cooperative learning approach to the traveling salesman problem, *IEEE transaction on evolutionary computation*, 1(1):53-66, 1997.
- [10] X.H. Shi, Y.C. Liang, H.P. Lee, C. Lub, Q.X. Wang, Particle swarm optimization-based algorithms for TSP and generalized TSP, *Information processing letters*, 103(2007): 169–176.
- [11] Xiaojun Zhou, Chunhua Yang, Weihua Gui, Initial version of state transition algorithm, International conference on digital manufacturing and automation (ICDMA), 2011: 644-647.
- [12] Xiaojun Zhou, Chunhua Yang, Weihua Gui, A new transformation into state transition algorithm for finding the global minimum, International conference on intelligent control and information processing(ICICIP), 2011: 674–678.